НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Теплоенергетичний факультет
Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:
Завідувач кафедри
_____ Олександр Коваль
«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Геометричне моделювання в інформаційних системах»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «Засоби пошуку інформаційних ресурсів у динамічному реєстрі»

Виконав:
студент IV курсу, групи ТР-61
Герасимчук Владислав Федорович _____

Керівник:
Старший викладач
Колумбет Вадим Петрович _____

Консультант:
_____

Рецензент:
_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Геометричне моделювання в інформаційних системах

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ
на дипломну роботу студенту
Герасимчуку Владиславу Федоровичу
(прізвище, ім’я, по батькові)

1. Тема роботи «Засоби пошуку інформаційних ресурсів у динамічному реєстрі»
керівник роботи Колумбет Вадим Петрович, старший викладач
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020р. **№ 1168-с**

2. Строк подання студентом роботи _____
3. Вихідні дані до роботи мова програмування JavaScript, серверна мова Node.js, середовище Visual Studio Code, мова опрацювання запитів GraphQL, реляційна база даних MySQL.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) проаналізувати проблему пошуку інформаційних ресурсів у динамічному реєстрі, розглянути засоби пошуку та оптимізовані алгоритми, розробити систему пошуку інформаційних ресурсів у динамічному реєстрі.
5. Перелік ілюстративного матеріалу зображення реалізованої системи засобів пошуку інформаційних ресурсів у динамічному реєстрі, зображення етапів інформаційного пошуку, видів пошуку, графік зростання кількості

студентів, на навчання яких вплинув карантин, схема обробки завантаження файлу сервером Node.js, зображення коректного тіла запиту на пошук документу у графічному представленні Apollo серверу, зображення частини GraphQL-запиту

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання "___"_____201___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	25.10.19	
2.	Вивчення та аналіз задачі	27.11.19-16.12.19	
3.	Розробка архітектури та загальної структури системи	17.12.19-27.01.20	
4.	Розробка структур окремих підсистем	28.01.20-06.04.20	
5.	Програмна реалізація системи	07.04.20-04.05.20	
6.	Оформлення пояснювальної записки	05.05.20-04.06.20	
7.	Захист програмного продукту	08.06.20	
8.	Передзахист	08.06.20	
9.	Захист		

Студент _____ _____Герасимчук В.Ф._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Колумбет В.П._____

АНОТАЦІЯ

Метою дипломної роботи є реалізація засобів пошуку інформаційних ресурсів у динамічному реєстрі. Роботу виконано на 69 аркушах, вона містить 3 додатки та перелік посилань на використані джерела з 35 найменувань, 28 рисунки. Система пошуку інформаційних ресурсів у динамічному реєстрі була створена для вирішення проблеми доступу до електронних документів. Вона повинна поліпшити, прискорити та оптимізувати процес навігації по ресурсам кафедри та допомога у роботі користувачам, сфера зайнятості яких пов’язана з електронним документообігом.

Ключові слова: інформаційно-пошукова система, GraphQL, PDF-файл, web-додаток, оптимізація процесу навігації, алгоритм Кнута-Морріса-Пратта.

ABSTRACT

The purpose of the thesis is the implementation of search tools for information resources in the dynamic register. The work is performed on 69 sheets, it contains 3 appendixes and a list of references to the sources with 35 items, 28 figures. The system of searching for information resources in the dynamic register was created to solve the problem of access to electronic documents. It should improve, accelerate and optimize the process of navigating the resources of the department and assistance to users whose employment is related to electronic document management.

Keywords: information retrieval system, GraphQL, PDF-file, web-application, navigation process optimization, Knut-Morris-Pratt algorithm.

ЗМІСТ

Перелік умовних позначень, скорочень і термінів 6
Вступ 7
1 Задача пошуку інформаційних ресурсів у динамічному реєстрі 9
2 Аналіз проблеми пошуку інформаційних ресурсів у динамічному реєстрі 13
2.1 Тенденції переходу до віддаленого навчання 14
2.2 Огляд подібних існуючих систем 15
3 Засоби розробки системи пошуку інформаційних ресурсів у динамічному реєстрі 16
3.1 Середовище розробки Visual Studio Code 16
3.2 Фреймворк React 18
3.3 Серверна платформа Node.js 19
3.4 Інструменти розробника Chrome DevTools 21
3.5 Архітекутра MVC 24
3.6 Середовище для виконання запитів GraphQL 25
3.7 Система управління реляційними базами даних MySQL 27
4 Опис програмної реалізації 29
4.1 Структура програмного забезпечення 29
4.2 Концептуальна модель 32
4.3 Внутрішня реалізація 37
5 Робота користувача з програмною системою 41
5.1 Системні вимоги 41
5.2 Робота користувача з програмним продуктом 42
Висновки 44
Список використаних джерел 45
Додаток 1 49
Додаток 2 51
Додаток 3 59

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API (англ. Application Programming Interface) — це сукупність засобів та правил, що вможливлюють взаємодію між окремими складниками програмного забезпечення або між програмним та апаратним забезпечення.

ІПС — інформаційно-пошукові системи.

REST (англ. Representational State Transfer) — підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів.

GraphQL (англ. Graph Query Language) — це мова запитів і маніпуляції даними з відкритим кодом для API і середовище виконання для обслуговування запитів з наявних даних.

VS Code (англ. Microsoft Visual Studio Code) — засіб для створення, редагування сучасних веб-застосунків і програм для хмарних систем.

PDF (англ. Portable Document Format) — відкритий формат файлу для представлення двовимірних документів.

Virtual DOM (англ. Virtual Document Object Model) — це концепція програмування, в якій ідеальне чи «віртуальне» представлення інтерфейсу користувача зберігається в пам’яті і синхронізується зі «справжнім» DOM за допомогою бібліотеки, такої як ReactDOM.

CPU (англ. Central Processing Unit) — функціональна частина комп'ютера, що призначена для інтерпретації команд.

CRUD (англ. Create Read Update Delete) — 4 базові функції управління даними «створення, зчитування, зміна і видалення».

MVC (англ. Model View Controller) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

ВСТУП

Нині цифровий світ росте дуже швидкими темпами. Розвиток інтернету залишає за собою великий обсяг онлайн-сервісів, цифрової економіки та електронної комерції, великий електронний документообіг. Зараз ми спокійно можемо сісти у таксі чи піти в магазин без гаманця, бо є електрона платіжна система, замовити техніку чи будь-який інший потрібний нам предмет не виходячи з дому, перевести кошти в одне натискання з телефону, побачити далеких родичів, не їдучи за 2000 кілометрів.

Всі види контенту переходять з аналогових, фізичних і статичних в цифрові, одночасно становляться мобільними та персональними. Разом з тим проблема електронного документообігу в організації стає дуже актуальною. Підприємці намагаються перевести бухгалтерську звітність в електронний формат чи навіть у додаток на телефоні, лікарі зробити єдиний реєстр пацієнтів, викладачі впорядкувати та структурувати учбовий план.

Світ намагається зробити все простішим за допомогою цифровізації, але в деяких випадках, як наприклад з документообігом, вирішення проблеми ускладнюється, за рахунок дуже великої кількості документів. Раніше роль сховища для документів грав архів, де всі папери зберігалися в алфавітному порядку, згідно з специфікації, тощо, і працівник архіву повинен був знати систему групування паперів. На сьогоднішній день проблема зберігання документів майже зникла завдяки хмарним середовищам чи накопичувачам, але з’явилась нова – оптимізований пошук.

Термін «інформаційний пошук» був вперше введений Кельвіном Муерсом в 1948 в його докторської дисертації.

Спочатку системи автоматизованого Інформаційного Пошуку, або інформаційно-пошукові системи (ІПС), використовувалися лише для пошуку наукової інформації і літератури. Багато університетів і публічні бібліотеки стали використовувати ІПС для забезпечення доступу до книг та журналів. Широке поширення ІПС отримали з появою мережі Інтернет і розвитком

Всесвітньої павутини. А згодом в оптимізований пошук стали потребувати і системи документообігу.

1 ЗАДАЧА ПОШУКУ ІНФОРМАЦІЙНИХ РЕСУРСІВ У ДИНАМІЧНОМУ РЕЄСТРІ

Моя мета реалізувати систему засобів пошуку інформаційних ресурсів у динамічному реєстрі за допомогою теоретичної літератури, наданої дипломним керівником. Система повинна виявляти в множині інформаційних ресурсів кафедри – ті, які задовольняють заздалегідь визначеним умовам пошуку або містять необхідні дані.

Взагалі, завданням інформаційного пошуку є знаходження відповідних (до пошукового запиту) інформаційних об'єктів, або документів серед доступного для пошуку матеріалу. Завдання для інформаційного пошуку задається у вигляді інформаційного запиту (query), який може містити слова, фрази чи речення або комбінацію їх.

У системах інформаційного пошуку прийнято говорити термінами “Запит” та “Об’єкт запиту”.

Запит - це формалізований спосіб вираження інформаційних потреб користувачем системи. Для вираження інформаційної потреби використовується мова пошукових запитів, синтаксис варіюється від системи до системи. Крім спеціальної мови запитів, сучасні пошукові системи дозволяють вводити запит на природній мові.

Об'єкт запиту - це інформаційна сутність, яка зберігається в базі автоматизованої системи пошуку. Незважаючи на те, що найбільш поширеним об'єктом запиту є текстовий документ, не існує ніяких принципових обмежень. Зокрема, можливий пошук зображень, музики та іншої інформації.

Пошук виконується в чотири етапи (Рис. 1):

1. Визначається власник і інформаційна потреба.
2. Формулюється запит.
3. Вилучається інформація з інформаційного масиву.

4. Ознайомлення з отриманою інформацією і оцінка результатів пошуку.

Рис. 1 – Етапи інформаційного пошуку

Наразі існує безліч методів пошуку, з них основні представлені на Рис. 2.

Рис. 2 – Види інформаційного пошуку

Адресний пошук

Пошук документів відбувається за допомогою формальних ознак, які в свою чергу зазначені у запиті. Для адресного пошуку повинні бути такі умови:

1. Точна адреса документа.
2. Точний порядок розташування документів.

Під адресами може бути веб-сервер або сторінка, бібліографічний запис або адреса зберігання документів у сховищі.

Семантичний пошук

Пошук документів здійснюється на основі змісту . Для семантичного пошуку повинні бути такі умови:

1. Зміст документу перекладається з природної мови на інформаційно-пошукову.
2. При складанні пошукового опису вказується додаткова умова.

Якщо порівнювати адресний і семантичний пошуки, то можна прогледіти, що об’єктом адресного пошуку є документ, якщо його сприймати як форму, а об’єктом семантичного пошуку документ з точки зору змісту.

Документальний пошук

Здійснюється пошук у сховищі інформаційно-пошукової системи або в базі даних. Є два види документального пошуку:

1. Бібліотечний, спрямований на знаходження первинних документів.
2. Бібліографічний, спрямований на знаходження відомостей про документи, представлених у вигляді бібліографічних записів.

Фактографічний пошук

Процес пошуку фактів, відповідних інформаційним запитом. До фактографічних даних відносяться відомості, витягнуті з документів, як первинних, так і вторинних і одержувані безпосередньо з джерел їх виникнення.

Система пошуку інформації також складається з двох компонентів: системи індексації та системи запитів. Перший з них відповідає за аналіз документів, завантажених з Інтернету, та створення індексів, які потім дозволяють здійснювати пошукові запити; а другий - видимий інтерфейс пошукової системи, тобто частина, з якою взаємодіють користувачі.

Якщо пошукова система здатна відповідати на питання настільки дивовижно за короткий проміжок часу, до якого ми звикли (як правило, частки секунди), це тому, що вони не досліджують Інтернет для користувачів у режимі реального часу (тобто як і коли запит робиться), а скоріше вони використовують індекс, який регулярно оновлюється (кілька разів на день).

Система, яка намагалася б досліджувати документи в Інтернеті послідовно і в режимі реального часу, щоб надати відповіді, не мала б найменшого сенсу. Дні або навіть тижні можуть пройти між запитанням та отриманням відповіді. Замість цього те, що робить пошукова система - це запитувати її внутрішні індекси.

Більшість систем пошуку інформації базуються на певній формі індексації. Каталог бібліотек насправді є своєрідним покажчиком, хоч часто і досить складним. Він посилає користувача на певні номери полиць (ті номери, які використовуються для розміщення книг та інших фізичних ресурсів інформації на полицях) або до певних файлів у всесвітній мережі. Аналогічно, покажчик на звороті книги посилає читача на номери сторінок.
Інші системи індексації відносяться не до фізично цілих предметів, ні до окремих сторінок, а до серії сторінок, які складають певну статтю - вони індексують статті з журналів та інші форми серійних публікацій. Сьогодні більшість цих систем є в Інтернеті, і багато з них складають дуже великі бази даних, що охоплюють сотні чи тисячі серіалів; багато також надають посилання на онлайн-версії деяких статей, які вони індексують. Поряд із звичайними типами бібліографічного елемента, такими як назва, ім'я автора, дескриптори предметів, багато систем містять реферати або короткі резюме деяких або всіх індексованих статей.

2 АНАЛІЗ ПРОБЛЕМИ ПОШУКУ ІНФОРМАЦІЙНИХ РЕСУРСІВ У ДИНАМІЧНОМУ РЕЄСТРІ

Нині багато підприємств і учбових закладів переносять вже діючі документи до електронної системи з технологіями документообігу. В цей же час недоліки, що існували в процесі документообігу можуть проявити свою дію в електронній системі, що може заплутати вже звичну структуру зберігання даних.

2.1 Тенденції переходу до віддаленого навчання

В останні місяці через пандемію COVID-19 в усіх підприємствах та державних установах, включаючи університети, постало нагальне питання електронного документообігу. У світі зріс попит на віддалену роботу (Рис. 2.1), що потягнуло за собою проблеми з швидким пошуком документів.

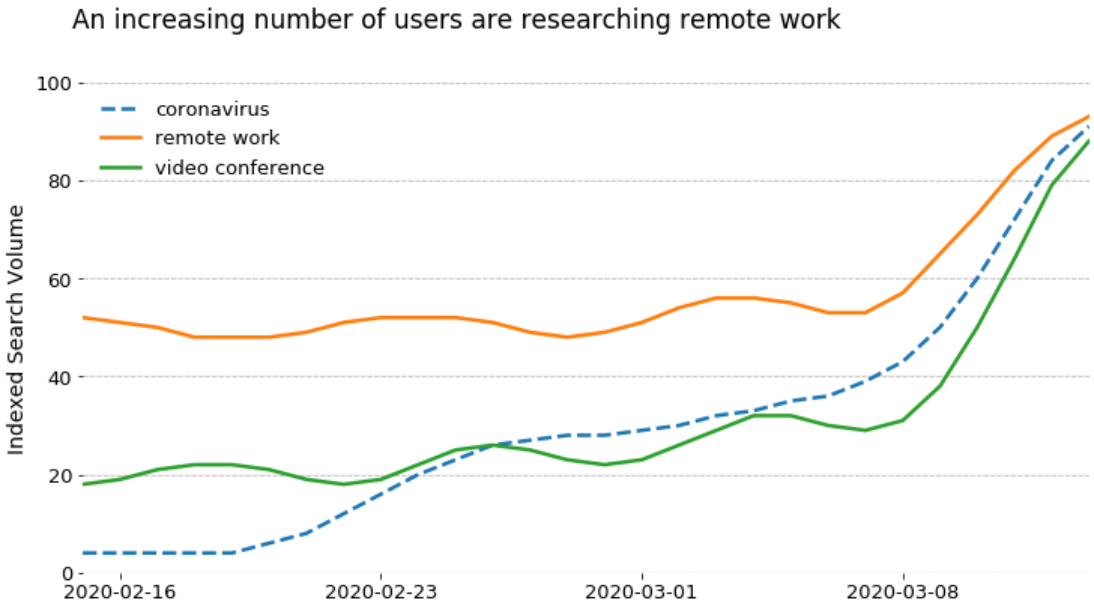

Рис. 2.1 – Графік збільшення попиту на віддалену роботу під час пандемії COVID-19

2.2 Огляд подібних існуючих систем

Розглядаючи електронний документообіг Національного технічного університету України “Київського політехнічного інституту імені Ігоря Сікорського” під час карантину, можна зрозуміти, яка величезна кількість документів потрібна чи є створеною за цей період.

На даний момент існує система під назвою ela.kpi.ua, що містить функціонал відповідальний за документообіг. Перелік дій даного ресурсу є достатнім для функціонування електронного реєстру ресурсів університету, але для того щоб прискорити пошук потрібної інформації й зробити навігацію більш зручною, спрямованою саме на потреби кафедри, було прийняте рішення створити автономну систему пошуку для інформаційних ресурсів кафедри.

Аналізуючи кількість документацію за період атестації – це відомості про кожну групу студентів, у період захисту дипломів – це дипломи та тези, у періоду вступу до магістратури - анкети для складання Єдиного Вступного Іспиту, а для вступу до бакалаврату університету потрібні табеля, які можуть будуть подаватися в електронному вигляду, якщо карантин продовжиться, а ще є учбові плани, затвердженні екзаменаційні запитання, контрольні роботи, тощо. З наведених прикладів видно, що електронних документів дуже побільшало за період пандемії. На Рис. 2.2 можна побачити, як зросла кількість школярів і студентів, на навчання яких вплинув карантин, і які були вимушені перейти до онлайн-навчання, а як наслідок, зросла електронна документація їх навчання.

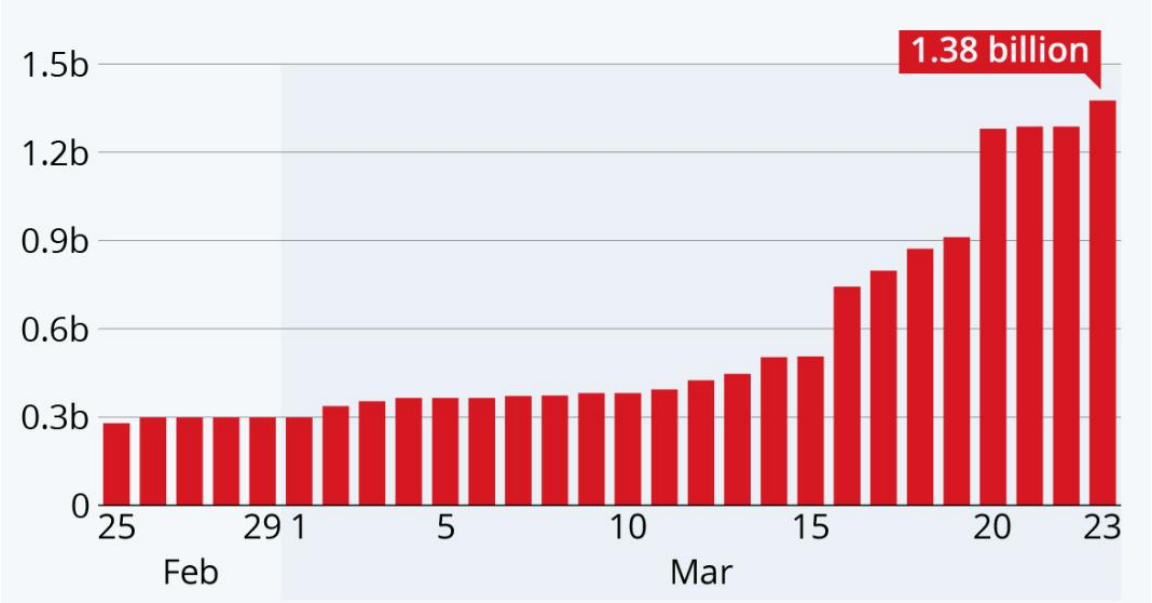

Рис. 2.2 – Графік зростання кількості студентів, на навчання яких вплинув карантин

Для вирішення проблеми швидкого доступу до будь-якого електронного документу з великої кількості можна застосувати систему пошуку. У системи пошуку є свої переваги та недоліки. Перевагами є те, що:

1. Результати передбачувані, пояснити їх досить просто.
2. Можна застосувати багато корисних функцій.
3. Ефективна обробка, оскільки багато документів можна усунути з пошуку.

Недоліками може бути:

1. Ефективність, яка залежить повністю від користувача.
2. Прості запити зазвичай погано працюють.
3. Складні запити важко обробляються.

3 ЗАСОБИ РОЗРОБКИ СИСТЕМИ ПОШУКУ ІНФОРМАЦІЙНИХ РЕСУРСІВ У ДИНАМІЧНОМУ РЕЄСТРІ

Для розробки системи засобів пошуку інформаційних ресурсів у динамічному реєстрі було використано високорівневу та багатопарадигмальну мову JavaScript, середовище сервера з відкритим кодом Node.js, середовище для виконання запитів GraphQL та його частина Apollo, модуль для зчитування PDF-файлів pdf-parse, JavaScript-фреймворк для створення користувацьких інтерфейсів React, базу даних з відкритим кодом MySQL, інструменти проксі-серверу бази даних MySQL – Prisma, набір інструментів для веб-розробників Chrome DevTools, бібліотека Bootstrap а також мову розмітки HTML5 та спеціальну мову стилю сторінок CSS.

3.1 Середовище розробки Visual Studio Code

Visual Studio Code - це редактор вихідного коду, розроблений Microsoft для Windows, Linux та macOS. Він надзвичайно швидкий і легкий редактор вихідного коду, який можна використовувати для перегляду, редагування, запуску та налагодження вихідного коду для додатків.

Visual Studio працює лише на операційній системі Windows та операційній системі Mac. У Visual Studio Code є багато переваг:

1. IntelliSense для мови програмування: IntelliSense - загальний термін, що використовується для різноманітних функцій редагування коду: автодоповнення коду, інформація про параметри, швидка інформація. Код VS IntelliSense надається для JavaScript, TypeScript, JSON, HTML, CSS, Less та Sass. Також можна додати розширення IntelliSense для інших мов, які не підтримуються за замовчуванням.

2. Visual Studio підтримує вбудований інтегрований термінал, починаючи з кореня відкритого проекту. Ця функція робить середовище дуже зручним, оскільки для виконання швидкого завдання з командним рядком нам не потрібно перемикати вікна або змінювати стан існуючого терміналу. PowerShell всередині цього інтегрованого середовища готовий виконувати загальновживані завдання за допомогою скорочень. Коли користувач починає вводити будь-яку з цих розпізнаваних команд, набір команд забезпечується IntelliSense допомогою і навіть запускає командний рядок для виконання завдань, як показано на Рис.3.1.

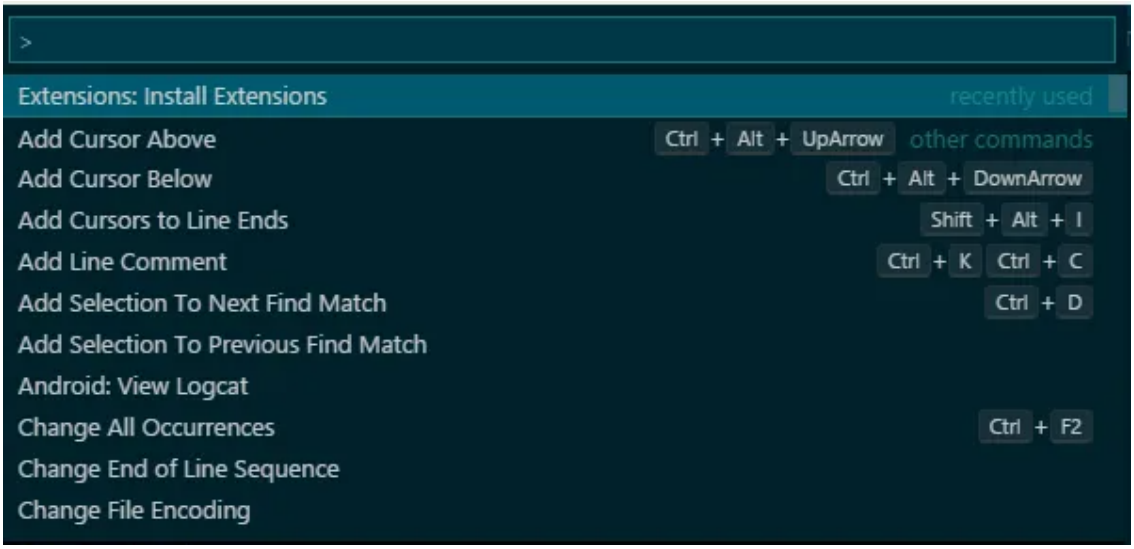

Рис. 3.1 – Запуск командного рядка у Visual Studio Code

3. Інтегрований контроль над версіями (вбудований Git): Код Visual Studio має вбудовану інтеграцію Git (Рис. 3.2), завдяки чому миттєво бачити зміни, які користувач вносить у свій проект. Ліворуч від бокової панелі ми можемо знайти піктограму Git, де ми можемо ініціалізувати Git, а також виконати декілька Git-команд, таких як: “commit”, “pull”, “push”, “rebase” та “publish”.

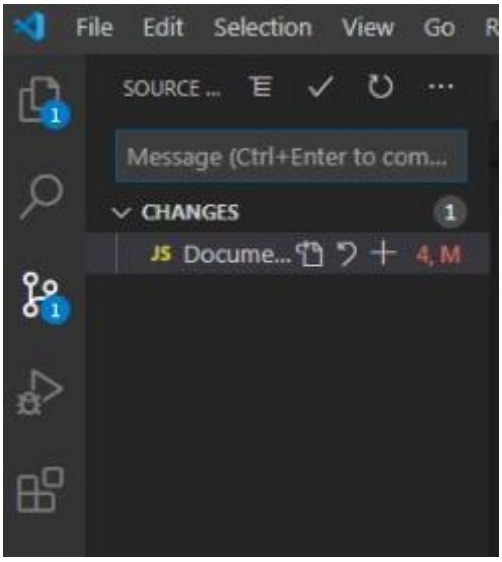

Рис. 3.2 – Панель змінених файлів

VS Code працює з будь-яким локальним або віддаленим сховищем Git і пропонує візуальне відображення для вирішення конфліктів до того, як код закомітиться.

3.2 Фреймворк React

React – це розширення синтаксису до JavaScript, що значно спрощує написання власних компонентів. Незважаючи на те, що React часто викликає суперечки, він може виявитися корисним у створенні великих обсягів додатків або спеціальних компонентів. Крім того, він надає розробникам React інформативні попередження та повідомлення про помилки, а також допомагає запобігти неконтрольованим вставкам коду.

React підвищує продуктивність і полегшує подальше обслуговування програмного коду. Розробники доповнили React можливістю повторного використання системних компонентів, а розробники визначають це як одну з найкращих особливостей React js.

Усі компоненти мають свою внутрішню логіку, що полегшує їх маніпулювання та визначення. Такий підхід забезпечує організований вигляд програми та полегшує підтримку та зростання бази даних коду.

Створюючи додаток з високим навантаженням, важливо врахувати, як структура вплине на загальну ефективність програми. Для вирішення проблеми команда розробників React представила Virtual DOM - на даний момент одна з переваг використання React для важких та динамічних програмних рішень. Як випливає з назви, це віртуальне представлення об'єктної моделі документа, тому всі зміни спершу застосовуються до віртуальної DOM, а потім обчислюється мінімальний обсяг необхідних DOM-операцій. У кінці реальне дерево DOM оновлюється відповідно, забезпечуючи мінімальний витрачений час. Цей метод гарантує кращу роботу користувачів та більш високу продуктивність додатків.

Я обрав React для створення системи засобів пошуку інформаційних ресурсів у динамічному реєстрі, бо працюю над додатком на одній сторінці та хочу зробити його швидким та зручним для користувачів. Враховуючи плюси і мінуси React js, його можна легко підсумувати двома словами: не ризикований та вдосконалений. Основна ідея цієї бібліотеки полягає в тому, щоб «створити масштабні програми з даними, які неодноразово будуть змінюватись з часом», і вона добре справляється з цим завданням. Він надає розробникам можливість роботи з віртуальним браузером (DOM), який набагато швидший і зручніший, ніж реальний.

3.3 Серверна платформа Node.js

Node.js - це середовище сервера з відкритим кодом, він працює на різних платформах (Windows, Linux, Unix, Mac OS X тощо) та використовує JavaScript на сервері.

Загальним завданням веб-сервера може бути відкриття файлу на сервері та повернення вмісту клієнту. В цьому завданні Node.js буде переможним рішенням. На прикладі PHP або ASP парсингу, як вони обробляють запит на файл:

1. Надсилає завдання файловій системі комп'ютера.

2. Зачекає, поки відкриється файлова система, і прочитає файл.
3. Повертає вміст клієнту.
4. Готовий прийняти наступний запит.

Ось як Node.js обробляє запит на файл:

1. Надсилає завдання файловій системі комп'ютера.
2. Готовий прийняти наступний запит.
3. Коли файлова система відкрилася і прочитала файл, сервер повертає вміст клієнту.

На прикладі обробки запиту Node.js стає видно, що він виключає очікування і просто продовжує наступний запит. Якщо брати життєвий випадок, можна описати, коли на сервер приходить запит на завантаження файлу (Рис. 3.3). Сервер приймає цей запит і поки він його обробляє (вертикальна стрілка праворуч), він також обробляє інші запити, перш ніж поверне відповідь на основний запит. Це і є концепція паралелізму, яка лягає в основу Node.js.

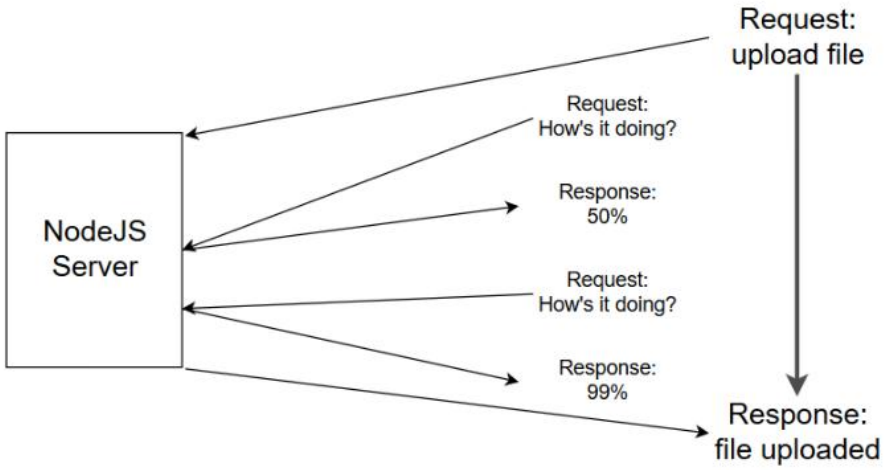

Рис.3.3 – Схема обробка завантаження файлу сервером Node.js

Node.js надає одно потокове, не блокуюче, асинхронне програмування, що дуже ефективно працює з пам'яттю. Node.js може генерувати динамічний вміст сторінки та може створювати, відкривати, читати, писати, видаляти та закривати файли на сервері. Також Node.js може збирати дані форми,

додавати, видаляти, змінювати дані у користувацькій базі даних. Особливістю Node.js є те, що файли повинні бути ініційовані на сервері, перш ніж мати якийсь ефект.

3.4 Інструменти розробника Chrome DevTools

Інструменти для розробників Google Chrome, також відомі як Chrome DevTools, - це інструменти для веб-розробки та налагодження, вбудовані прямо в браузер. Вони надають розробникам більш глибокий доступ до своїх веб-додатків та браузера. Користувач може зробити все: від тестування вікна та перегляду на мобільному пристрої до редагування веб-сайту на ходу та навіть вимірювання продуктивності всього веб-сайту чи окремих активів. Щоб використовувати останню версію інструментів для розробників, користувач може скористатися Chrome Canary - експериментальною версією Chrome. Chrome DevTools має багато панелей. Нижче представлено опис функціональності кожної панелі.

Панель Elements використовується для вибору та редагування будь-яких HTML-елементів на сторінках. Дозволяє вільно маніпулювати DOM та CSS. Вкладка (Рис. 3.4) містить дві кнопки: вибір елемента за допомогою курсору і перемикання в режим вибору пристроїв, вона стане в нагоді при розробці адаптивних інтерфейсів, мобільних версій сайтів або для тестування сторінок з різним розширенням монітора.

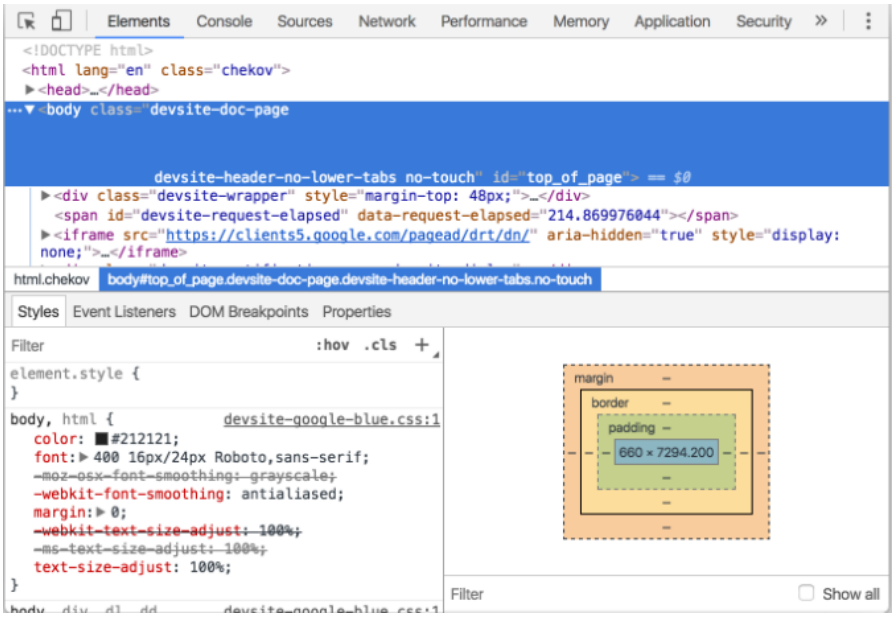

Рис 3.4 – Вкладка Elements

Панель Console необхідна для логування діагностичної інформації в процесі розробки або взаємодії з JavaScript на сторінці. Також всі помилки в JavaScript коді, будуть виводиться тут із зазначенням файлу і конкретного місця в ньому де сталася помилка. Так само в консоль можна виводити XHR запити. Є можливість зберігати логи в окремий файл.

Інструмент Sources є середовищем, де ми можемо подивитися всі файли підключені на нашій сторінці. Ми можемо подивитися їх вміст, відредагувати код, скопіювати його або зберегти змінений файл, як новий файл. Дану вкладку можна використовувати і як повноцінний редактор коду.

Вкладка Network дозволяє моніторити процес завантаження сторінки і всіх файлів які завантажуються. Її зручно використовувати для оптимізації завантаження сторінок і моніторингу запитів. На панелі відображається таблиця всіх запитів до даних і файлів, над нею розташовуються кнопки для фільтрації потрібних користувачу запитів, очищення таблиці або включення / відключення запису запитів, кнопки управління відображенням таблиці.

Під таблицею вказано кількість всіх запитів, загальна кількість завантажених даних, загальний час завантаження всіх даних, час

завантаження і побудови DOM дерева і час завантаження всіх ресурсів, які впливають на відображення цієї сторінки. Можливості вкладки Network:

1. Можливість відключити кешування або встановлення обмеження пропускної здатності.
2. Отримання докладної таблиці з інформацією про кожного запиті.
3. Фільтрація і пошук по всьому списку запитів.

Панель Performance (Рис. 3.5) відображає таймлайн використання мережі, виконання JavaScript коду і завантаження пам'яті. Після початкової побудови графіків таймлайн, будуть доступні дані про виконання коду. У вкладці є можливість ознайомлення з часом виконання окремих частин коду, вибору окремого проміжку на часовій шкалі та ознайомлення з тим, які процеси відбувалися в цей момент. На вкладці Performance можливо:

1. Зробити запис, щоб проаналізувати кожну подію, яка сталася після завантаження сторінки або взаємодії з користувачем.
2. Переглянути FPS, завантаження CPU і мережеві запити в області Overview.
3. Натиснути по події в діаграмі, щоб подивитися деталі про неї.
4. Змінити масштаб таймлайну, щоб зробити аналіз більш ефективним.

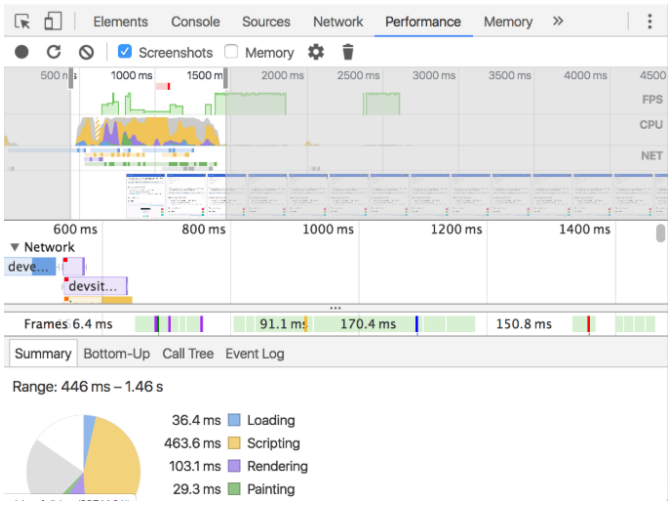

Рис. 3.5 – Вкладка Performance

Ключові можливості Memory панелі - виправлення проблем з пам'яттю та профілювання CPU при роботі з JavaScript.

Вкладка Application призначена для інспектування та очищення всіх завантажених ресурсів, включаючи IndexedDB або Web SQL бази даних, куків, кеша додатків, зображень, шрифтів і таблиць стилів. Можливості вкладки Application:

1. Швидке очищення сховищ і кеша.
2. Інспектування і управління сховищами, базами даних і кешем.
3. Інспектування і видалення файлів cookie.

На вкладці Security можна ознайомиться з протоколом безпеки при його наявності і переглянути дані про сертифікат безпеки, якщо він є. Цей інструмент використовується для налагодження проблем змішаного контенту, проблем сертифікатів та інше.

3.5 Архітекутра MVC

MVC розшифровується як Model-View-Controller. Це архітектура або модель дизайну програмного забезпечення, що робить створення величезних додатків простим. Він не належить до конкретної мови програмування. MVC (Рис. 3.6) це концепція, яку ви можете використовувати при створенні будь-якого типу додатків або програмного забезпечення будь-якою мовою програмування.

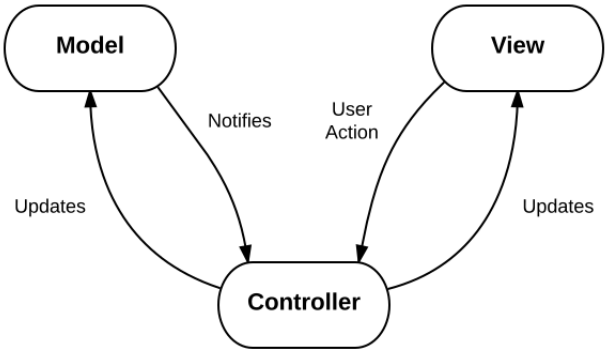

Рис. 3.6 – Робота архітектури MVC

Обрана архітектура містить три різні частини. Різні частини - це модель, вид та контролер. Кожна з них відіграє важливу роль у розробці додатків.

Модель працює безпосередньо з базою даних. Вона не повинна бути з'єднаною з користувальницьким інтерфейсом або обробкою даних. У реальному сценарії користувач просто буде використовувати модель для отримання, вставлення, оновлення та видалення даних зі своєї бази даних.

Вид - це інтерфейс користувача, на якому клієнт / користувач може виконувати деякі дії. Він містить HTML, CSS, JS, XML або будь-яку іншу мову розмітки, яку ми можемо використовувати для створення інтерфейсу користувача. Він також містить код для відображення даних, які він отримує з програми.

Єдина функція Виду - це показувати клієнту / користувачеві дані на Інтерфейсі користувача та реагувати на події.

Контролер - це частина, в якій ми обробляємо дані після отримання запиту від Виду і до того, як будь-що оновиться в нашій базі даних за допомогою Моделі.

3.6 Середовище для виконання запитів GraphQL

GraphQL - це мова запитів для API та середовище для виконання цих запитів із наявними даними. GraphQL надає повний і зрозумілий опис даних у API, дає клієнтам можливість запитувати саме те, що їм потрібно, і нічого більше та надає потужні інструменти для розробників.

Програми, що використовують GraphQL, є швидкими та стабільними, оскільки вони керують отриманими даними, а не сервером.

GraphQL запити мають доступ не лише до властивостей одного ресурсу, але й плавно слідують за посиланнями між ними. У той час як типові API REST вимагають завантаження з декількох URL-адрес, API GraphQL отримують усі дані, що потрібні додатку в одному запиті. Додатки,

що використовують GraphQL, можуть бути швидкими навіть у повільних мобільних мережних з'єднаннях. На Рис. 3.7 показано схему запиту клієнта за допомогою GraphQL і відповіді сервера.

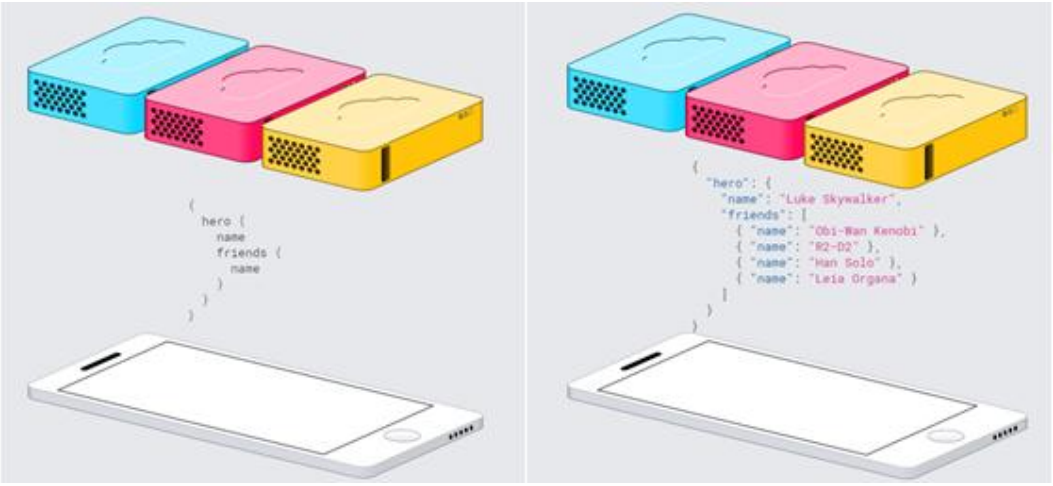

Рис. 3.7 – Запит клієнта за допомогою GraphQL і відповідь сервера.

API для GraphQL організовані у вигляді типів та полів, а не кінцевих точок. Користувач має змогу отримувати доступ до всіх можливостей даних з однієї кінцевої точки. GraphQL використовує типи, щоб додатки запитували лише те, що можливо, і забезпечує явні помилки. Програми можуть використовувати типи, щоб уникнути написання коду для ручного парсингу.

На Рис. 3.8 зліва показано приклад запиту на GraphQL, а справа схема можливих запроси.

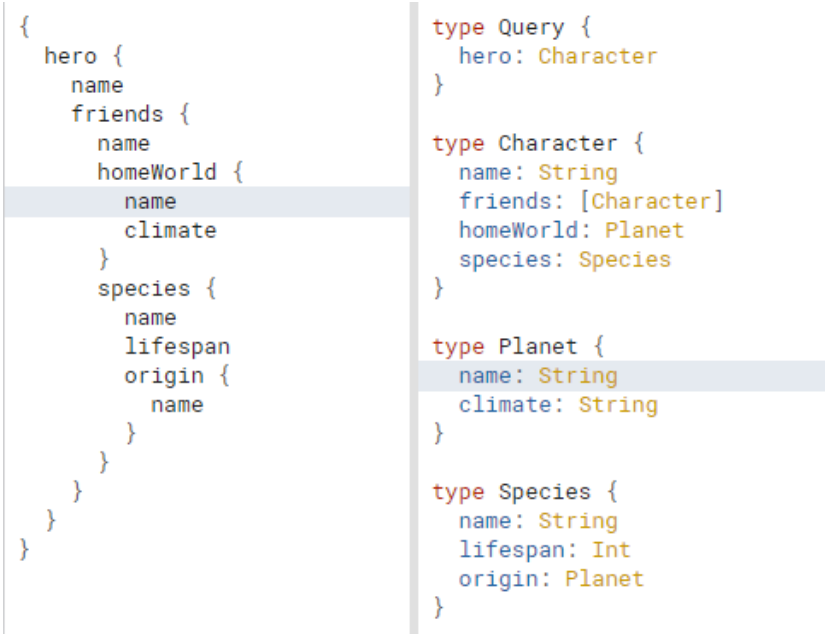

Рис. 3.8 – Приклад запиту на GraphQL та схема можливих запитів.

3.7 Система управління реляційними базами даних MySQL

MySQL - це система управління реляційними базами даних з відкритим кодом, заснована на структурованій мові запитів (Structured Query Language). MySQL доступний у всіх основних операційних системах, включаючи Windows, Linux та Solaris.

Вона, як і інші реляційні бази даних, зберігає дані в таблицях, стовпцях та рядках. Кожен запис визначається унікальним ідентифікатором. Основою MySQL завжди були продуктивність та надійність бази даних. MySQL був розроблений та оптимізований для веб-розробок; це, мабуть, найпоширеніша база даних, яка використовується при розгортанні веб-серверів.

MySQL виявився успішним, оскільки він простий у використанні, простий в установці та використовує мову запиту, яку легко зрозуміти. Наприклад, команда SHOW DATABASES відобразить список доступних баз даних у MySQL (Рис. 3.9).

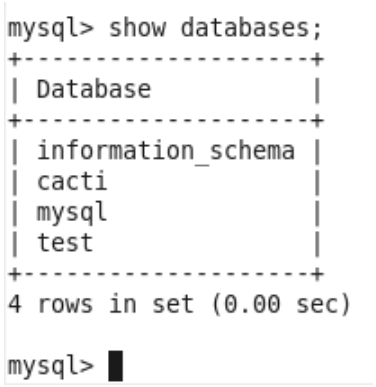

Рис. 3.9 – Приклад роботи команди “SHOW DATABASES”.

У MySQL є багато переваг:

1. Зниження загальної вартості власності: MySQL - одна з найпопулярніших систем управління базами даних з відкритим кодом, яка дозволяє керувати реляційною базою даних. Оскільки MySQL з відкритим кодом, користувач може використовувати MySQL безкоштовно та має можливість налаштувати його вихідний код відповідно до потрібних вимог.
2. Портативність: MySQL - це крос-платформенний сервер баз даних. Він може працювати на різних платформах, таких як Linux, Solaris та Windows. MySQL підтримує багато форм з різними мовами, такими як C, C ++, Node.js, PHP, PERL, JAVA, Python тощо.
3. Швидкий розвиток і цілодобова робота: MySQL має гарантію тривалості роботи 24x7 і пропонує широкий спектр високодоступних рішень, включаючи спеціалізовані кластерні сервери. У MySQL є дуже велике співтовариство розробників, яке випускає регулярне оновлення.
4. Безпека даних: MySQL визнається в усьому світі найбезпечнішою та надійною системою управління базами даних, яка використовується у популярних веб-додатках.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

При підборі архітектурного рішення для системи засобів пошуку інформаційних ресурсів у динамічному реєстрі вибір впав на MVC (Model-View-Controller) через ряд причин. По-перше, MVC надає можливість швидкої реалізації програмного продукту завдяки вибраному архітектурному шаблону. По-друге, завдяки MVC легко співпрацювати та працювати разом з декількома розробниками, а так як система пошуку створювалась паралельно системі парсингу та додавання даних іншим розробником, то вибір шаблону архітектури був очевидний.

4.1 Структура програмного забезпечення

Програмно-апаратна частина сервісу реалізована на програмній платформі node.js за допомогою нового підходу до архітектури мережевих протоколів GraphQL та його частини Apollo. Також для створення серверної частини був використаний модуль для зчитування PDF-файлів pdf-parse.

В системі пошуку інформаційних ресурсів GraphQL описує як запитувати дані, і, в основному, використовується клієнтом для завантаження даних з сервера. GraphQL дозволяє користувачу точно вказати, які дані йому потрібні, полегшує агрегацію даних з декількох джерел та використовує систему типів для опису даних.

Apollo Server допомагає системі створити прошарок GraphQL на існуючій API та REST. Apollo - це галузева реалізація GraphQL, що забезпечує рівень графу даних, який з'єднує програми з хмарою. У системі Apollo компоненти просто декларують свої вимоги до даних за допомогою GraphQL, а Apollo отримує потрібні дані в потрібне місце та запобігає помилкам. Він представляє всі джерела даних організації як один підключений графік даних, який завжди актуальний. Система пошуку

інформаційних ресурсів має змогу легко переглядати все, що є в наявності, приєднуватися до даних у різних джерелах та отримувати результати у потрібній формі та на будь-якій платформі.

Засоби пошуку здійснюються за допомогою інструментів проксі-серверу бази даних MySQL - Prisma, який перетворює базу даних в API зручне для використання в GraphQL. Для зберігання документів була обрана база даних MySQL.

Prisma - це окремий компонент системи пошуку інформаційних ресурсів, який розгортається перед базою даних MySQL і генерує GraphQL API. На основі створеної моделі даних Prisma створила готовий до використання GraphQL API, відкривши схему CRUD GraphQL.

Для завантаження та парсингу PDF-об'єктів та їх заголовків в системі використано модуль pdf-parse.

Клієнтська сторона користувальницького інтерфейсу була реалізована засобами мови програмування JavaScript, мовою тегів HTML, спеціальна мова стилю сторінок CSS, бібліотекою Bootstrap та фреймворком React.

Для відображення опцій пошуку та додавання документів був використаний React.

Система пошуку була розроблена на чистому node.js без застосування модулів, для її реалізації було застосовано алгоритм Кнута-Морріса-Пратта (скорочено алгоритм КМП). КМП є один з алгоритмів пошуку рядка, що шукає входження слова W у рядку S, використовуючи просте спостереження, що коли відбувається невідповідність, то слово містить у собі достатньо інформації для того, щоб визначити, де наступне входження може початися, таким чином пропускаючи кількаразову перевірку попередньо порівняних символів.

Серверна частина back-app містить в собі серверний код, який відповідає за збереження файлу, підключення до бази даних, пресинг та відповіді клієнту. Prisma відповідає за сутності у базі даних та за з’єднання з

нею. Папка src містить у собі частини парсингу та за відповіді до запитів (Рис.4.1).

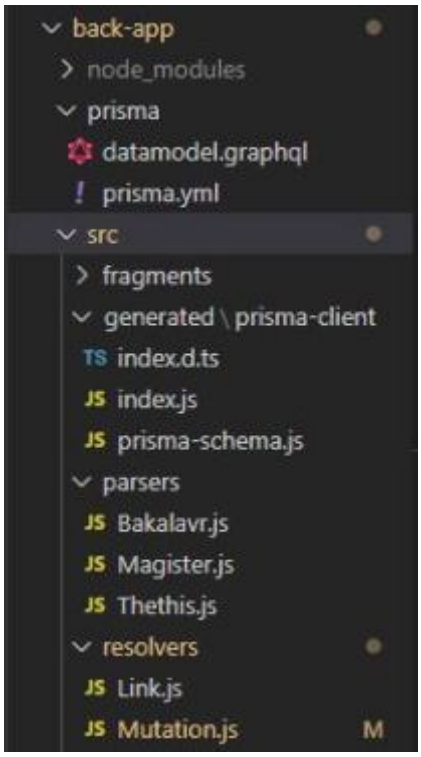

Рис 4.1 – Серверна частина back-app.

Структура клієнтської частини зображена на Рис. 4.2.

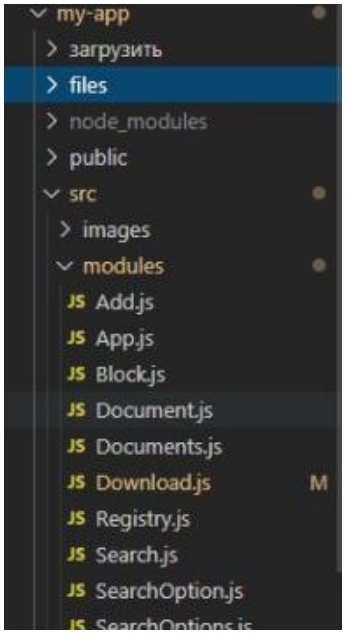

Рис 4.2 – Структура клієнтської частини.

4.2 Концептуальна модель

Проаналізувавши потреби та предметну область системи пошуку інформаційних ресурсів була створена концептуальна модель, яка містить головну сутність Ресурс та дочірні:

1. Тип ресурсу.
2. ID.
3. Автор.
4. Науковий керівник.
5. Заголовок.
6. Спеціалізація.
7. Спеціальність.
8. Напрям.
9. УДК.
10. Зміст.

Тип ресурсу відповідає за тип документу, який зберігається в реалізованій системі.

У таблиці Автор зберігається прізвище, ім’я та ім'я по батькові людини, яка написала певну роботу.

У таблиці Науковий керівник зберігається прізвище, ім’я та ім'я по батькові людини, яка перевірила певну роботу.

У таблиці Заголовок зберігається інформація про теми певних робіт.

У таблиці Спеціалізація зберігається спеціалізація навчання людини, яка написала певну роботу. У таблиці Спеціальність зберігається спеціальність навчання людини, яка написала певну роботу.

У таблиці Напрям міститься дані про напрям підготовки людини, яка написала певну роботу. У таблиці УДК міститься універсальна десяткова класифікація певних робіт, тобто бібліотечна класифікація документів.

У таблиці Зміст міститься контент різних робіт.

Коли система пошуку інформаційних ресурсів у динамічному реєстрі робить запит POST, Apollo Server приймає запити POST з тілом JSON. Дійсний запит повинен містити або запит, або ім'я операції (або обидва, у випадку названого запиту), і може містити змінні. На Рис 4.3 зображено приклад коректного тіла запиту на пошук конкретного документу.

Рис 4.3 – Приклад коректного тіла запиту на пошук документу у графічному представленні Apollo серверу.

З прикладу POST-запиту можна виділити такі частини, як тип операції, назву операції та змінні визначення, які позначені рамками фіолетового кольору на Рис. 4.4.

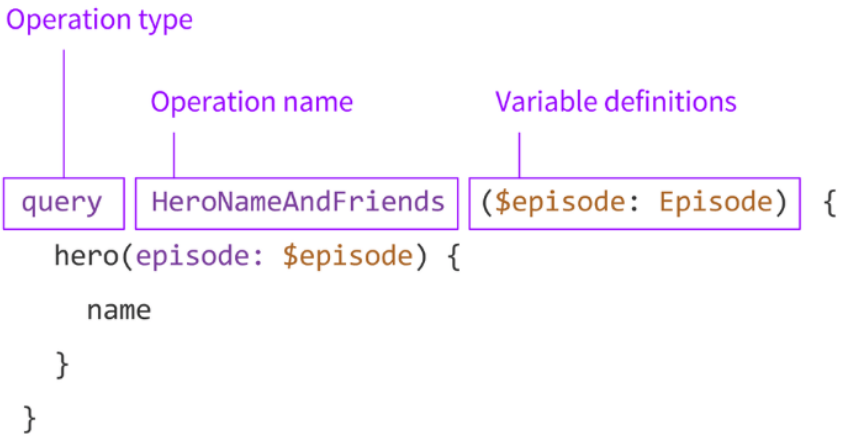

Рис. 4.4 – Частини GraphQL-запиту.

Тип операції: це або запит, мутація, або підписка. В цій частині описано, який тип операції користувач намагаєтеся зробити. Хоча всі вони схожі в мові, вони мають дещо різні режими виконання на специфічно сумісному сервері GraphQL.

Назва операції: для налагодження та реєстрації на стороні сервера запитам надавались імена. Таким чином, при ре факторингу системи у мережевих журналах чи на GraphQL-сервері, є можливість легко знайти цей запит у своїй кодовій базі за назвою, а не намагатися розшифрувати вміст. Цю назву можна порівняти з назвою функції на будь-якій мові програмування.

Змінні визначення: коли користувач надсилає запит на сервер GraphQL, у системі пошуку інформаційних ресурсів наявні динамічні частини, що змінюються між запитами, в той час як фактичний документ запиту залишається однаковим. Це змінні запиту користувача. Оскільки GraphQL статично типізований, він фактично може підтвердити, що користувач переходить в потрібні змінні.

На Рис. 4.5 зображено відповідь Apollo серверу на POST-запит з Рис. 4.3.


```
{
  "data": {
    "postResource": {
      "id": "ckb1qia9lkzif09681gztaxuk",
      "author": [
        {
          "fullName": "1"
        }
      ],
      "subAuthor": [
        {
          "fullName": "2"
        }
      ],
      "title": {
        "title": "3"
      },
      "direction": {
        "code": "4"
      },
      "profession": {
        "profession": "5"
      },
      "specialization": {
        "specialization": "6"
      },
      "udc": {
        "udc": "7"
      },
      "content": {
        "content": "8"
      }
    }
  }
}
```


Рис. 4.5 – Відповідь Apollo серверу на POST-запит.

Також у графічному представленні ми можемо побачити концептуальну модель системи пошуку інформаційних ресурсів у динамічному реєстрі. Модель містить список запитів, мутацій та детальна інформація щодо типів. На Рис. 4.6 можна побачити список запитів до кожної сутності та детальну інформацію щодо типів до сутності “Ресурси”.

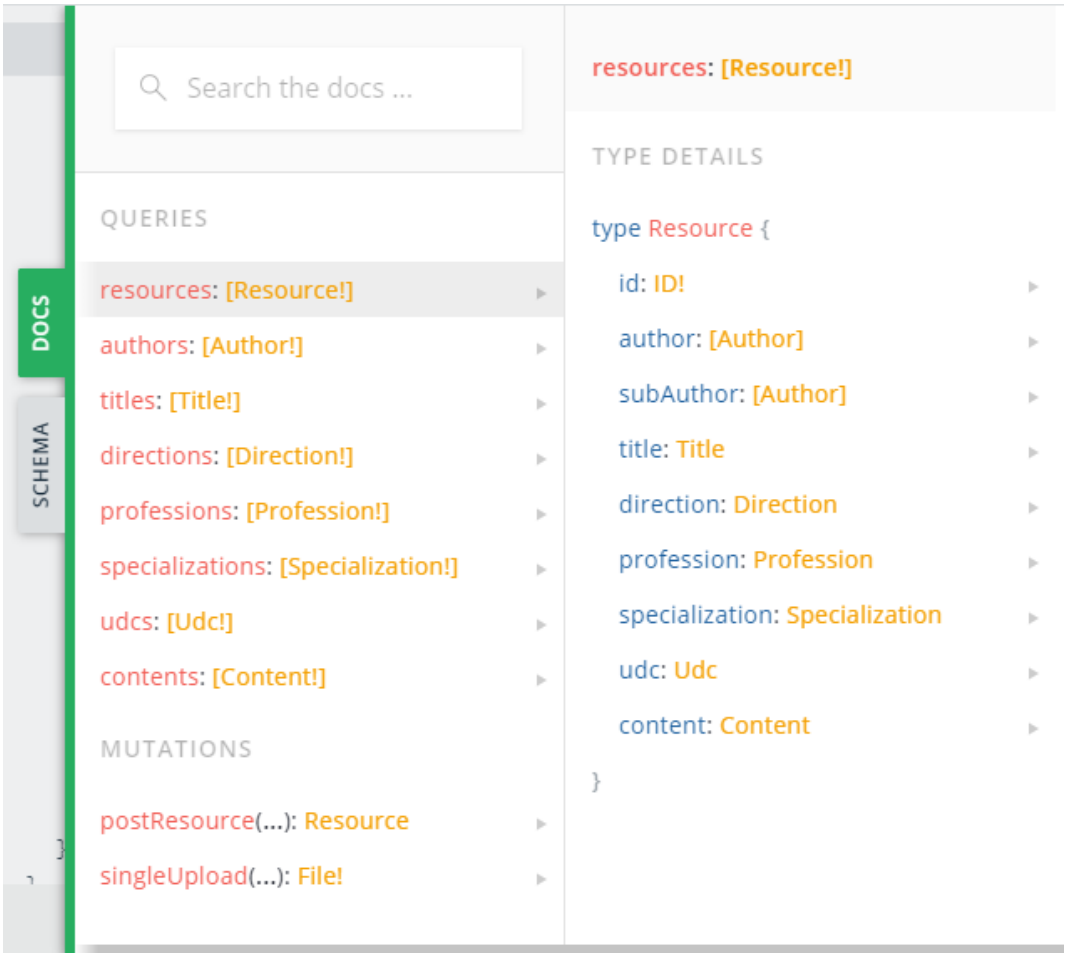

Рис. 4.6 – Список запитів до кожної сутності та детальна інформація щодо типів до сутності “Ресурси”.

При натисканні на поле id зі списку детальної інформації щодо типів сутності “Ресурси” з’являється інформація про примітивний тип, це показано на Рис. 4.7.

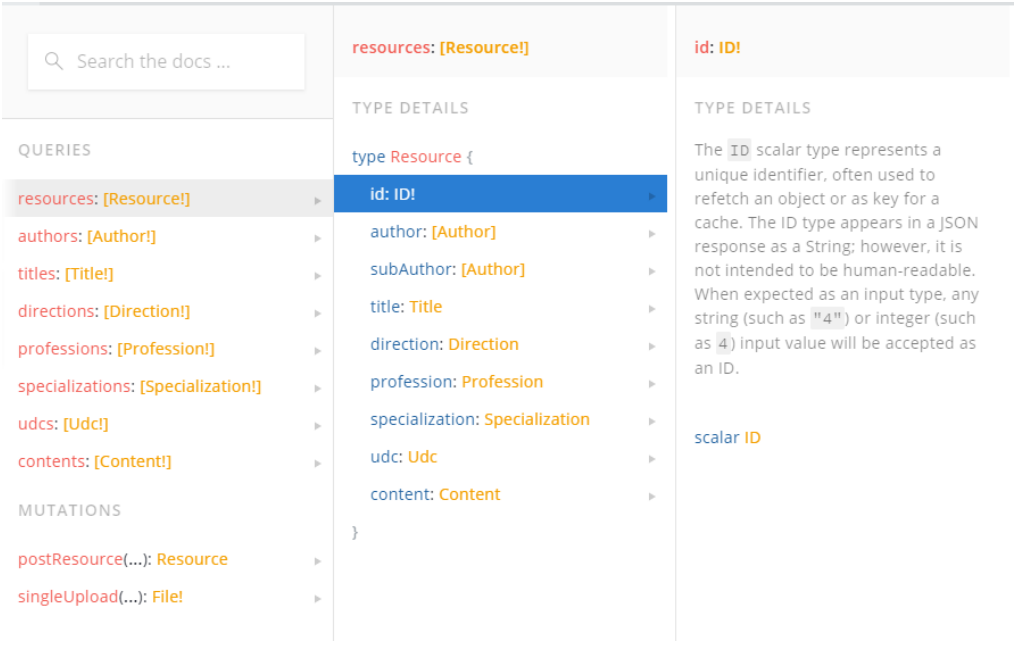

Рис. 4.7 – Інформація про примітивний тип id.

При натисканні на будь-яке інше поле зі списку детальної інформації щодо типів сутності “Ресурси” з’являється інформація про відповідний тип, але він вже не є примітивним, бо створений у ході реалізації системи пошуку. Інформацію про тип “Title” показано на Рис. 4.8.

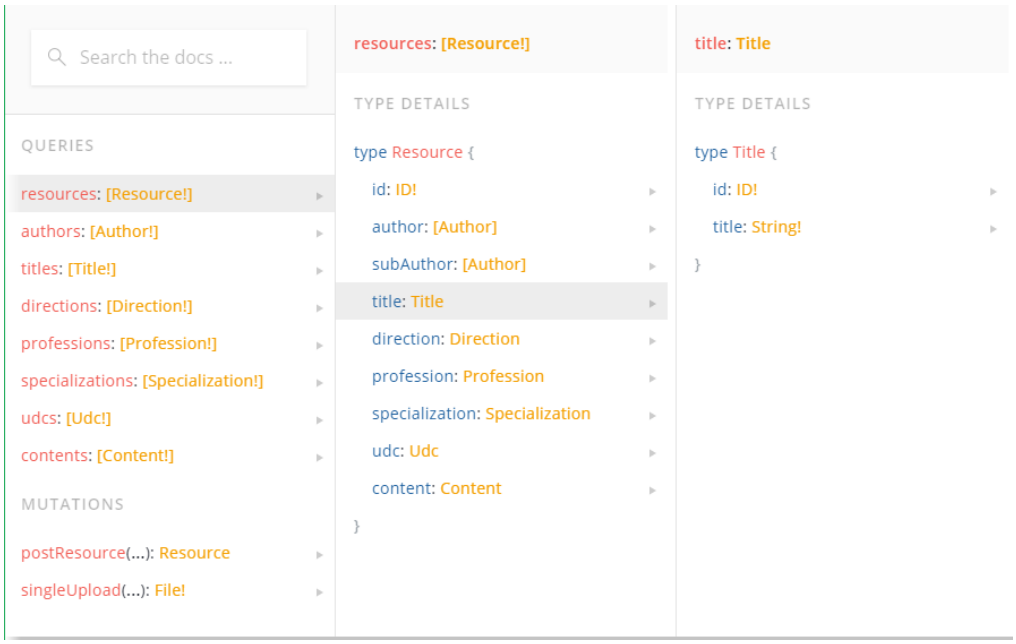

Рис. 4.8 – Інформація про Title.

Також у графічному представленні Apollo серверу можна побачити позначення шаблону “назва_типу!”, це означає, що це значення повинно обов’язково бути заповненим, приклад зображено на Рис. 4.9.

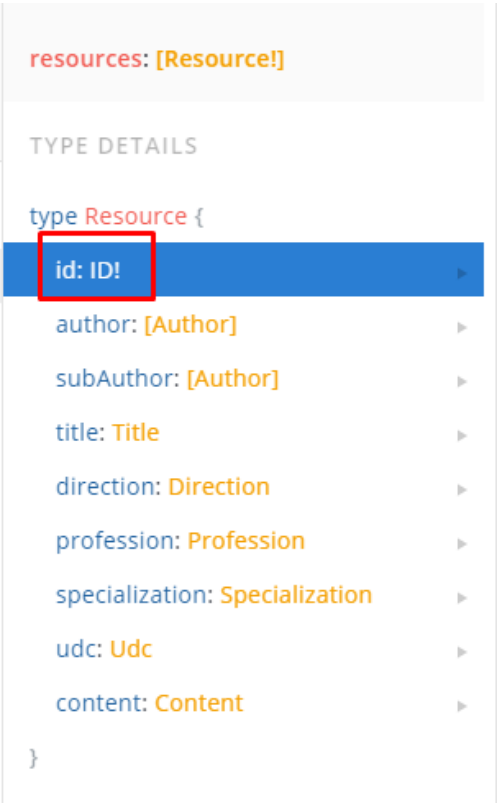

Рис. 4.9 – Значення, яке повинно обов’язково бути заповненим.

4.3 Внутрішня реалізація

Користувач на клієнтській стороні вказує тип документу, завантажує його, потім цей pdf-документ завантажується на сервер, як це показано на Рис. 4.10. Сервер розпаршує цей документ по частинах, які містять певний визначаючий контент, наприклад ім’я чи заголовок, та зрівнює ці частини для визначення типу документу. Таким чином, відбувається перевірка для подальшого запобігання завантаження некоректних файлів до таблиці типів та з’являється можливість шукати документ згідно визначаючим контент частинам. Якщо система знайшла всі частини у певному типі документу, то

вона заповнює таблицю ресурс і зв’язані з нею таблиці згідно з розпаршеними даними.

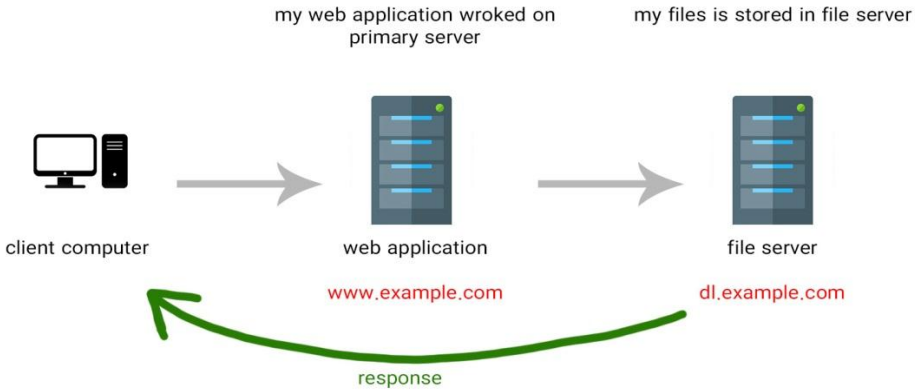

Рис. 4.10 – Завантаження PDF-файлу до серверу.

Коли користувач шукає документ з усіх завантажених, то пошук відбувається за алгоритмом Кнута-Морріса-Пратта.

Часто доводиться стикатися зі специфічним пошуком, так званим пошуком рядків (пошуком в рядку). Нехай є деякий текст T і слово W. Необхідно знайти перше входження цього слова в зазначеному тексті. Ця дія типова для будь-яких систем обробки текстів.

Найбільш типовим застосуванням такого завдання є документальний пошук: заданий фонд документів, що складається з послідовності бібліографічних посилань, кожне посилання супроводжується «дескриптором», що вказує тему відповідного посилання. Треба знайти деякі ключові слова, що зустрічаються серед дескрипторів. Міг би мати місце, наприклад, запит «Програмування» і «Java». Такий запит можна трактувати наступним чином: чи існують статті, які мають дескрипторами «Програмування» і «Java».

Пошук рядка формально визначається наступним чином. Нехай заданий масив T з N елементів і масив W з M елементів, причому $0 < M \leq N$. Пошук рядка виявляє перше входження W в T, результатом будемо вважати

індекс “i”, який вказує на перший з початку рядка (з початку масиву T) збіг зі словом. Наприклад, користувачеві потрібно знайти всі входження зразка W = abaa в текст T = abcabaabcabca (Рис. 4.11). Зразок входить в текст тільки один раз, зі здвигом S = 3, індекс i = 4.

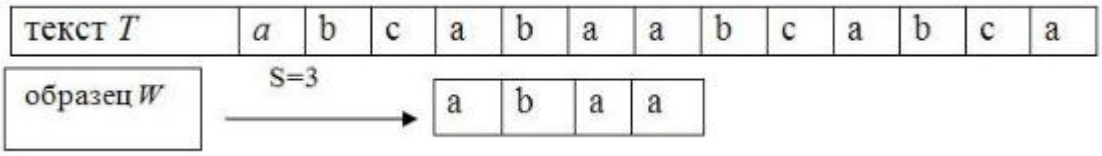

Рис. 4.11 – Приклад пошуку входження зразка в текст.

Алгоритм КМП-пошуку фактично вимагає лише близько N порівнянь навіть в найгіршому випадку. Наприклад, на Рис. 4.12 символи, які зазнали порівняння, підкреслені.

Рис. 4.12 – Приклад КМП-пошуку.

Після часткового збігу початкової частини зразку W з відповідними символами рядка T ми фактично знаємо пройдену частину рядка і можемо «вирахувати» деякі відомості (на основі самого зразку W), за допомогою яких потім швидко просунемося по тексту.

Ідея КМП-пошуку - при кожній розбіжності двох символів тексту і зразку зразок зсувається на всю пройдену відстань, так як менші зсуви не можуть привести до повного збігу.

Особливості КМП-пошуку:

1. Потрібно близько (N + M) порівнянь символів для отримання результату.
2. Схема КМП-пошуку дає справжній виграш тільки тоді, коли невдачі передувало деяке число збігів. Лише в цьому випадку зразок зсувається більш ніж на одиницю.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Метою створення системи засобів пошуку інформаційних ресурсів у динамічному реєстрі було вирішення проблеми швидкого доступу до будь-якого електронного документу, які зберігаються у великій кількості.

5.1 Системні вимоги

Для продуктивного використання системи засобів пошуку інформаційних ресурсів у динамічному реєстрі потрібні такі системні характеристики:

1. 64-розрядна архітектура.
2. Ядро версії 3.10 або вище.
3. Один з таких варіантів операційної системи:
    1. Windows 7/10.
    2. Ubuntu 14.04 / 15.10.
    3. CentOS 6.x.
    4. Red Hat Enterprise Linux (RHEL) 7.x.
    5. Debian 7.7.
4. 4 або більше процесорів / ядер.
5. Принаймні 16 ГБ оперативної пам’яті.
6. Принаймні 25 Гб місця на диску.
7. Відкриті порти для вхідного трафіку TCP: 8800 (консоль адміністратора), 8080 (реєстр), 8081 (веб-сайт).
8. Доступ до Інтернету безпосередньо чи через проксі.

5.2 Робота користувача з програмним продуктом

Щоб користуватися даним продуктом необхідно перейти у браузері до ресурсу на якому буде розташована користувацька частина програми або відкрити директорію в якій розташований програмний код та в консолі запустити виконання команди “npm start” та зачекати відкриття ресурсу у браузері за замовчуванням.

Виконання даної команди є процесом прослуховування порту під номером 3000, на якому працює локальний сервер, що відповідає за графічну частину представлення програмного продукту, яка в свою чергу обмінюється даними з серверною частиною продукту, відповідальною за обробку та зберігання даних.

Програма надає можливість обрати категорію та опції (Рис. 5.1), які необхідні для пошуку по ресурсам та конкретизації пошукових даних.

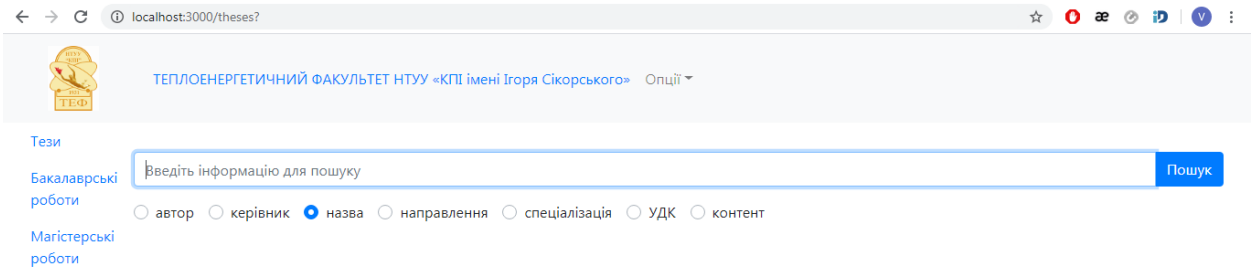

Рис. 5.1 – Вибір категорії та опцій пошуку

Після обрання необхідної категорії та опцій пошуку (Рис. 5.2) одержуємо результати певної категорії.

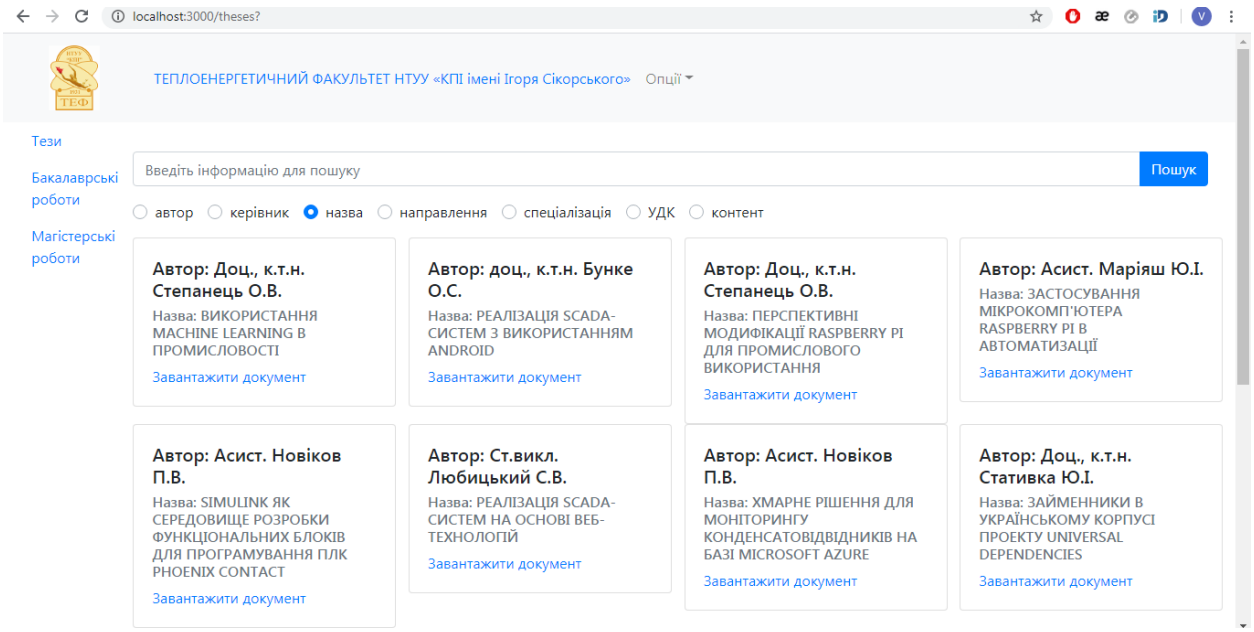

Рис. 5.2 – Конкретизація опцій пошуку

Виконавши пошук по заданим опціям в певній категорії отримуємо результат, який відповідає заданим параметрам (Рис. 5.3).

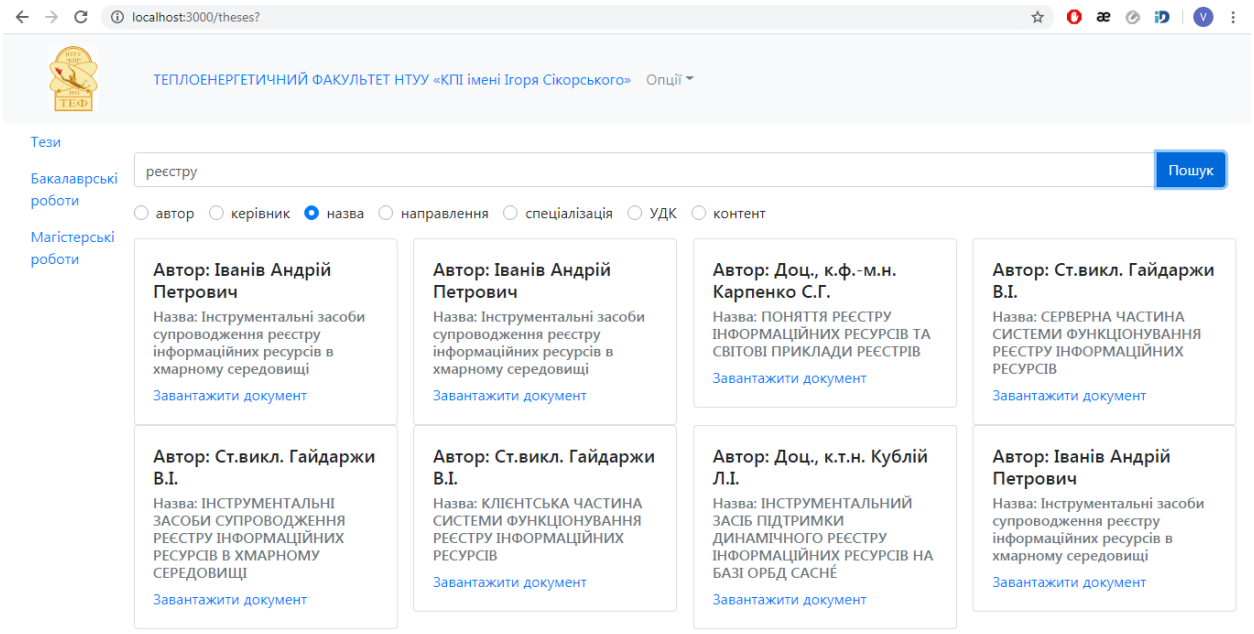

Рис. 5.3 – Результат пошуку

ВИСНОВКИ

У ході даної роботи було створено систему засобів пошуку інформаційних ресурсів у динамічному реєстрі, яка може поліпшити, прискорити та оптимізувати процес навігації по ресурсам кафедри та допомогти у роботі користувачам, сфера зайнятості яких пов’язана з електронним документообігом.

Для створення системи пошуку інформаційних ресурсів були використані такі інструменти, як мова програмування JavaScript, серверна платформа Node.js, фреймворк React.js, система управління базами даних MySQL.

Створена система матиме змогу робити оптимізований пошук та дасть можливість вибирати опції пошуку, згрупованому по автору, по назві ресурсу чи по їхньому змісту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інформаційно-пошукові системи [Електронний ресурс] – Режим доступу до ресурсу: https://ppt-online.org/169281.
2. Інформаційний пошук [Електронний ресурс] – Режим доступу до ресурсу:https://znaimo.com.ua/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B8%D0%B9_%D0%BF%D0%BE%D1%88%D1%83%D0%BA.
3. Інтерфейс користувача [Електронний ресурс] – Режим доступу до ресурсу: https://www.uk.w3ki.com/human_computer_interface/index.htm.
4. REST [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/REST.
5. Алгоритмы поиска в строке [Електронний ресурс] – Режим доступу до ресурсу: https://m.habr.com/ru/post/111449/.
6. Алгоритм Кнута — Морріса — Пратта [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D0%BD%D1%83%D1%82%D0%B0_%E2%80%94_%D0%9C%D0%BE%D1%80%D1%80%D1%96%D1%81%D0%B0_%E2%80%94_%D0%9F%D1%80%D0%B0%D1%82%D1%82%D0%B0.
7. Создание средства поиска [Електронний ресурс] – Режим доступу до ресурсу: https://signatov.com/sozdanie-sredstva-poiska/.
8. What is MySQL? [Електронний ресурс] – Режим доступу до ресурсу: https://www.atlantic.net/what-is-mysql/.
9. Інформаційний пошук [Електронний ресурс] – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/%D0%86%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D1%96%D0%B9%D0%BD%D0%B8%D0%B9_%D0%BF%D0%BE%D1%88%D1%83%D0%BA.
10. Проблеми електронного документообігу та шляхи їх вирішення [Електронний ресурс] – Режим доступу до ресурсу:

http://magazine.faaf.org.ua/problemi-elektronnogo-dokumentoobigu-ta-shlyahi-ih-virishennya.html.

11. Положення про Міністерство цифрової трансформації України від 18 вересня 2019 [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/856-2019-п#n12.
12. Information Retrieval Systems [Електронний ресурс] – Режим доступу до ресурсу: https://www.sciencedirect.com/topics/computer-science/information-retrieval-systems.
13. TOP 10 ADVANTAGES OF USING REACT.JS [Електронний ресурс] – Режим доступу до ресурсу: https://da-14.com/blog/its-high-time-reactjs-ten-reasons-give-it-try.
14. Node.js Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/nodejs/nodejs_intro.asp.
15. JavaScript [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/JavaScript.
16. What is Visual Studio Code and its advantages [Електронний ресурс] – Режим доступу до ресурсу: https://blog.webnersolutions.com/visual-studio-code/.
17. User interface [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/User_interface.
18. Обзор всех инструментов разработчика Chrome DevTools [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/company/simbirsoft/blog/337116/.
19. What is MVC? Advantages and Disadvantages of MVC [Електронний ресурс] – Режим доступу до ресурсу: https://www.interserver.net/tips/kb/mvc-advantages-disadvantages-mvc/.
20. MySQL Products [Електронний ресурс] – Режим доступу до ресурсу: https://www.mysql.com/products/.

21. MySQL Advantages and Disadvantages [Електронний ресурс] – Режим доступу до ресурсу: https://www.techstrikers.com/MySQL/advantages-and-disadvantages-of-mysql.php.
22. Алгоритм Кнута-Морриса-Пратта (КМП) [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/307220/.
23. Basics Tutorial - Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.howtographql.com/basics/0-introduction/.
24. Introduction to GraphQL [Електронний ресурс] – Режим доступу до ресурсу: Basics Tutorial - Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.howtographql.com/basics/0-introduction/.
25. ECMAScript 2015 (ES6) та вище [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/uk/docs/es6/.
26. Why Prisma? [Електронний ресурс] – Режим доступу до ресурсу: https://www.prisma.io/docs/understand-prisma/why-prisma.
27. How Apollo Server processes GraphQL operations [Електронний ресурс] – Режим доступу до ресурсу: https://www.apollographql.com/docs/apollo-server/data/resolvers/.
28. pdf-parse [Електронний ресурс] – Режим доступу до ресурсу: https://www.npmjs.com/package/pdf-parse.
29. Build fast, responsive sites with Bootstrap [Електронний ресурс] – Режим доступу до ресурсу: https://getbootstrap.com/.
30. Why React-Bootstrap? [Електронний ресурс] – Режим доступу до ресурсу: https://react-bootstrap.github.io/getting-started/why-react-bootstrap/.
31. HTML Styles - CSS [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/html/html_css.asp.
32. About npm [Електронний ресурс] – Режим доступу до ресурсу: https://docs.npmjs.com/about-npm/.
33. MVC для веб: проще некуда [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/181772/.

34. Живые и неживые коллекции в JavaScript [Електронний ресурс] – Режим доступу до ресурсу: http://htmlbook.ru/blog/zhivye-i-nezhivye-kollekcii-v-javascript.
35. A production-ready GraphQL layer over any backend [Електронний ресурс] – Режим доступу до ресурсу: https://www.apollographql.com/server/.

Додаток 1

Засоби пошуку інформаційних ресурсів
у динамічному реєстрі

Специфікація

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61_61104 20Б 81-1

Аркушів 1

Київ 2020

Позначення	Найменування	Відмітки
Документація		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61	Опис програмного модулю	Основні компоненти
УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61	serverPart.index.js	Основні компоненти

Додаток 2

Засоби пошуку інформаційних ресурсів
у динамічному реєстрі

Опис програмного модулю

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61_61104 20Б 81-2

Аркушів 8

Київ 2020

АНОТАЦІЯ

Додаток містить опис системи засобів пошуку інформаційних ресурсів у динамічному реєстрі.

В додатку виконуються такі функції:

— Збереження документів та їх форматування;

— Відділення та збереження метаданих з документів;

— Пошук ресурсу за категорією.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ 54
ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ 55
ОПИС ЛОГІЧНОЇ СТРУКТУРИ 56
ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ 57
ВИКЛИК І ЗАВАНТАЖЕННЯ 58

ЗАГАЛЬНІ ВІДОМОСТІ

У цьому додатку описано систему пошуку інформаційних ресурсів у динамічному реєстрі.

Модулі системи описані в ДОДАТКУ 3.

Систему було створено за допомогою високорівневої мови JavaScript, середовища серверу з відкритим кодом Node.js, середовища для виконання запитів GraphQL та його частини для графічного відображення API Apollo, модулю для зчитування PDF-файлів pdf-parse, JavaScript-фреймворку для створення користувацьких інтерфейсів React, бази даних з відкритим кодом MySQL, інструментів проксі-серверу бази даних MySQL – Prisma, набіру інструментів для веб-розробників Chrome DevTools, бібліотеки Bootstrap а також мови розмітки HTML5, спеціальної мови стилю сторінок та середовища розробки Visual Studio Code.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Систему пошуку інформаційних ресурсів у динамічному реєстрі можна використовувати як автономну систему документообігу. Ця система допоможе проводити пошук швидше та краще. Завдяки реалізованій системі поліпшиться, прискориться та оптимізується процесу навігації по ресурсам кафедри, що допоможе у роботі користувачам, сфера зайнятості яких пов’язана з електронним документообігом.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Реалізована система орієнтована на документообіг в період захисту дипломних робіт, а саме на магістерські та бакалаврські роботи, а також на тези.

Для функціонування системи потрібно встановити всі необхідні модулі та запустити клієнтську частину командою npm start й серверну частину командою node index.js.

ВИКОРИСТОВУВАНІ ТЕХІЧНІ ЗАСОБИ

Було створено клієнт-серверну систему яка функціонує наступним чином. База данних дає відповідь на запит одержаний Express сервером й оброблений GraphQL, відповідь отримана клієнською частиною обробляєть Apollo-клієнтом, та передається до React.js додатку який генерує графічне представлення відповіді.

Система функціонує на операційних системах, таких як Windows7, Windows8 та Windows10.

ВИКЛИК І ЗАВАНТАЖЕННЯ

Щоб користуватися даним продуктом необхідно перейти у браузері до ресурса на якому буде розташована користувацька частина програми. Обрати тип роботи по якому буде відбуватися пошук, задати опції пошуку та пошукові данні.

Додаток 3

Засоби пошуку інформаційних ресурсів
у динамічному реєстрі

Текст програмного модулю

УКР.НТУУ”КПІ”_ТЕФ_АПЕПС_ТР61_61104 20Б 81-3

Аркушів 11

Київ 2020

```
const express = require('express');
const { ApolloServer, gql } = require('apollo-server-express');
const { prisma } = require('./src/generated/prisma-client')
const typeDefs = require('./src/schema');
const Query = require('./src/resolvers/Query')
const Mutation = require('./src/resolvers/Mutation')

const resolvers = {
  Query,
  Mutation,
}

const server = new ApolloServer({
  typeDefs,
  resolvers,
  context: request => {
    return {
      request: request.req,
      prisma,
    }
  },
})

const app = express();
server.applyMiddleware({ app });

app.listen({ port: 4000 }, () =>
  console.log('Now browse to http://localhost:4000' + server.graphqlPath)
);
const fs = require('fs');
const pdf = require('pdf-parse');
const { fullResource } = require("../fragments/Resource")
```

```
const { parseThethis } = require("../parsers/Thethis")
const { parseMagister } = require("../parsers/Magister")
const { parseBakalavr } = require("../parsers/Bakalavr")

function postResource(parent, args, context, info) {
  const { author, subAuthor, title, direction, profession,
specialization, udc, content } = args

  return context.prisma.createResource({
    author: {
      create: {
        fullName: author
      }
    },
    subAuthor: {
      create: {
        fullName: subAuthor
      }
    },
    title: {
      create: {
        title
      }
    },
    direction: {
      create: {
        code: direction
      }
    },
    profession: {
      create: {
        profession
      }
    },
    specialization: {
```

```
            create: {
              specialization
            }
          },
          udc: {
            create: {
              udc
            }
          },
          content: {
            create: {
              content
            }
          }
        }).$fragment(fullResource)
      }

      async function createTheFile(file) {
        return new Promise(resolve => {
          //ToDo remove file from this, understand why
          const { createReadStream, filename, mimetype } = file

          const readFileStream = createReadStream()
          const                  readWriteStream                  =
fs.createWriteStream(`./uploadedFiles/${filename}`)
          readFileStream.pipe(readWriteStream)
          readWriteStream.on('finish', () => {
            resolve()
          })
        })
      }

      function singleUpload(parent, args, context, info) {
        return args.file.then(async file => {
          await createTheFile(file)
```

```
        const { createReadStream, filename, mimetype } = file

        //TODO rewrite
        let result
        if (args.type == "THETHIS") {
          result                    =                    await
parseThethis(`./uploadedFiles/${filename}`)
        } else if (args.type == "BAKALAVR") {
          result                    =                    await
parseBakalavr(`./uploadedFiles/${filename}`)
        } else if (args.type == "MAGISTER") {
          result                    =                    await
parseMagister(`./uploadedFiles/${filename}`)
        }
        if (args.type == "THETHIS") {
          result.forEach(async res => {
            const  {  author,  subAuthor,  title,  direction,
profession, specialization, udc, content } = res
            const request = {}
            if (author) {
              request.author = {
                create: {
                  fullName: author
                }
              }
            }
            if (subAuthor) {
              request.subAuthor = {
                create: {
                  fullName: subAuthor
                }
              }
            }
            if (title) {
              request.title = {
```

```
    create: {
      title
    }
  }
}
if (direction) {
  request.direction = {
    create: {
      code: direction
    }
  }
}
if (profession) {
  request.profession = {
    create: {
      profession
    }
  }
}
if (specialization) {
  request.specialization = {
    create: {
      specialization
    }
  }
}
if (udc) {
  request.udc = {
    create: {
      udc
    }
  }
}
if (content) {
  request.content = {
```

```
                create: {
                  content
                }
              }
            }
            await context.prisma.createResource({ ...request
}).$fragment(fullResource)
          })
        } else {
          const { author, subAuthor, title, direction,
profession, specialization, udc, content } = result
          const request = {}
          if (author) {
            request.author = {
              create: {
                fullName: author
              }
            }
          }
          if (subAuthor) {
            request.subAuthor = {
              create: {
                fullName: subAuthor
              }
            }
          }
          if (title) {
            request.title = {
              create: {
                title
              }
            }
          }
          if (direction) {
            request.direction = {
```

```
            create: {
              code: direction
            }
          }
        }
        if (profession) {
          request.profession = {
            create: {
              profession
            }
          }
        }
        if (specialization) {
          request.specialization = {
            create: {
              specialization
            }
          }
        }
        if (udc) {
          request.udc = {
            create: {
              udc
            }
          }
        }
        if (content) {
          request.content = {
            create: {
              content
            }
          }
        }
        await    context.prisma.createResource({    ...request
}).$fragment(fullResource)
```

```
    }

    return file;
  });
}

module.exports = {
  postResource,
  singleUpload
}
const fs = require('fs')
const pdf = require('pdf-parse')

function parseThethis(path) {
  let dataBuffer = fs.readFileSync(path);
  const regUDC = new RegExp(/\nУДК /gu)

  return pdf(dataBuffer).then(function (data) {
    const works = data.text.split(regUDC)
    const UDCs = []
    const author = []
    const subAuthor = []
    const title = []
    const content = []

    const regSpace = ' '
    const regEnter = '\n'
    const regSmallLetter = new RegExp(/\p{Ll}/gu)
    const regInitials = new RegExp(/\p{Lu}\.\p{Lu}\./gu)
    const regAllUpperLetterInWord = new
RegExp(/\p{Lu}\s{0,}\n\p{Lu}(\p{Ll}{1,}|\s)/gu)
    regAllUpperLetterInWord.lastIndex = 0;
    const regFirstContentWord = new
RegExp(/(^|\s)\p{Lu}(\p{Ll}{1,}|\s)/gu)
```

```
        for (let i = 2; i < works.length; i++) {
          const workSplittedSpace = works[i].split(regSpace)
          const workSplittedEnter = works[i].split(regEnter)

          UDCs.push(workSplittedSpace[0].trim())
          author.push(workSplittedEnter[1].trim())
          subAuthor.push(workSplittedEnter[2].trim())

          const enterTitlePart = []
          for (let j = 3; !regSmallLetter.test(workSplittedEnter[j]) && j < workSplittedEnter.length; j++) {
            enterTitlePart.push(workSplittedEnter[j])
          }
          title.push(enterTitlePart.join('').trim())

          const start = regAllUpperLetterInWord;
          const end = "Перелік посилань";
          const idxStart = works[i].search(start);
          const idxEnd = works[i].indexOf(end);
          content.push(works[i].slice(idxStart + 1, idxEnd));
        }
        const res = []
        for (let i = 0; i < author.length; i++) {
          res.push({
            author: author[i],
            author: subAuthor[i],
            title: title[i],
            udc: UDCs[i],
            content: content[i]
          })
        }

        return res
```

```
  })
}

module.exports = {
  parseThethis
}
```